GeneXus™

Grow thru knowledge

GeneXus Overview

Latest Update: 2012

GeneXus international
www.genexus.com

MONTEVIDEO - URUGUAY	Av. Italia 6201 - Edif. Los Pinos, P1	(598) 2601 2082
	Ruta 8 Km. 17.500 - Edif. @3 Of. 205	(598) 2518 2870
CHICAGO - USA	1143 W Rundell PL, Suite 200	(1 312) 836 9152
SÃO PAULO - BRASIL	Rua Samuel Morse 120 Conj. 141	(55 11) 5502 6722
CIUDAD DE MÉXICO - MÉXICO	Calle Leibnitz N° 20, desp. 801	(52 55) 5255 4733
TOKYO - JAPAN	Tk Gotanda Bldg. 303-2	(81 3) 5793 5481

Introduction

GeneXus is an intelligent tool developed by Artech aimed at assisting analysts and users throughout the lifecycle of applications.

The design and prototype are done and tested on a Windows NT/2000/XP/7 environment. Once the prototype is fully approved by users, the database and the application programs for the production environment are generated and/or maintained in a fully automatic way.

The core idea underlying GeneXus is the automation of everything that may be automated: data and design normalization, the generation and maintenance of the database and the application programs. This prevents analysts from spending time in routine and tedious tasks, leaving them free to fully concentrate on the one thing programs will never be able to do: **understand the issues faced by the user**.

As a byproduct, GeneXus offers rigorous, self-sufficient and constantly updated documentation.

The aim of this document is to provide readers with information on GeneXus and the problems that it solves.

Contents of the sections below:

- The theoretical question: this chapter presents a description comparing traditional system development methods and incremental development.
- An incremental development implementation: GeneXus.
- Unique features of GeneXus.
- Who the users of GeneXus?

The Theoretical Question

Traditional Development Methodologies and Associated Problems

The traditional method for developing applications is based on a core principle: **it is possible to build a stable data model of the business**. Based on this principle, the first task performed is the analysis of data, where reality is studied in an abstract way to obtain a product data model of the business. The second task is the design of the database. It is very simple to design a database starting from a known data model.

Once reality has been studied from the point of view of data, the same is done from the point of view of functions (functional analysis). It would be desirable that the study of reality resulted in a functional specification that depended solely on this reality. However, in the most used methodologies, a functional specification referred to the database files (or to the data model entities, which is essentially equivalent) is obtained.

Once the database and the functional specification have been obtained, the next step is the implementation of functions, for which there are traditionally several options (3rd or 4th generation languages, generators, and interpreters).

However, all these forms of implementation have a common problem: they are based on the principle stated above: **it is possible to build a stable data model of the business**, and this principle is **false**.

It is actually **impossible** to build, in an abstract way, a fully detailed and objective data model of the business, because nobody knows the business as a whole.

For this reason it is necessary to resort to multiple interlocutors, with each one of them projecting their own subjectivity on the model. A consequence of this is that during the lifecycle of the application, the model undergoes changes.

But even in an ideal situation, where all needs are fully accounted for, and where it would be therefore be possible to define an optimum database, the model would have to change to encompass the evolution of the business.

All this would be of little importance if the functional specification and the database were independent. However, since the functional specification refers to the database, the inevitable modifications in the latter determine the need for (manual) modifications in the former.

The most important consequence of this is that maintenance costs are extremely high: most companies working with the conventional methods agree that 80% of the resources theoretically allocated to development are actually used to maintain already implemented applications.

In the case of large applications the situation is even worse: maintenance starts much before implementation, which creates a hyperlineal increase of development costs against the size of the project.

Due to the difficulties in determining and propagating the consequences of changes made to the database to the processes in this context, instead of making the necessary changes, the option is frequently to introduce redundant files, with the consequent degrading of system quality and the increase of maintenance costs.

Knowledge-based Development and Incremental Methodology

In the last few years there has been a lot of talk about **Knowledge Management** in the industry and many things have been included under this label, which are very different from Knowledge-based Development as we describe it here.

The industry has usually used the term to refer to ways of organizing and/or accessing knowledge to be used in a traditional way by human beings. This is an updated version, using currently available technology, of books (and one that is enormously useful for humanity): by reading a book we access certain knowledge and, in our minds, we reason on this knowledge, which ultimately results in actions. Intelligent text searchers, which have become available in the last few years, make this knowledge increasingly accessible to more human beings.

As a general rule, this knowledge is not “understandable” for a machine and is, therefore, not operable. Additionally, as human beings are able to reasonably (up to a certain extent) deal with ambiguity and even with inconsistency, this knowledge is not always rigorous.

It is then important to define the concept of knowledge that we will use in our Knowledge-based Development. This knowledge meets the following conditions:

- It is rigorous
- It may be represented objectively
- It is operable

Finding a new way to solve the problem of system development requires replacing the basic principle stated above: assuming that **it is not possible to build a stable data model of the business** and, using instead an **incremental philosophy and creating a Knowledge-based Development**. An incremental approach seems natural: not dealing with large issues, but rather solving small problems as they appear.

What is the impact of this type of scheme on maintenance costs?

If the previously described methodologies were to be used with this approach, the impact on cost would be very high: the data model would be constantly modified and maintenance costs would be even higher than the ones described above.

The following is clear though: the database is not known, but each user, knows very well the data views they use everyday.

These data views may be of various types: screens, dialogs, process flows, lists, etc., which form the external appearance of the application: what is tangible for the user.

How can the knowledge of these views help in creating a data model?

Can the issue be turned into a logical/mathematical problem? If this were possible, logic and mathematics would provide a wide range of resources to help solve it automatically and, as a consequence, the analyst's job would be highly simplified. It is interesting to point out the following: if the database were known, it should be possible to derive from it the data views of the different users. That is to say, the database must fulfill all the views known. It may be demonstrated that, given a set of user data views, there is always a minimum database that fulfills it, which is also unique. At this stage, the problem has become a logical/mathematical problem and it is then necessary to solve it in order to find that database.

How Is This Theory Implemented?

The knowledge existing in the user views is captured and systematized in a knowledge base (all this is done automatically). The main characteristic of this knowledge base, which distinguishes it from the traditional data dictionaries, is its inference capacity: it is expected that, at any moment, it may be possible to retrieve from this knowledge base both the elements that have been placed in it and any other that may be inferred from them.

If this objective is met, the database and the application programs become deterministic transformations of this database and this permits to:

- Generate them automatically
- Determine the impact of changes made to the data and processes on the user data views and to propagate such changes generating:
  - the programs needed to convert data;
  - the application programs affected by the changes;
  - those application programs that have not been affected by the changes, but that may now be replaced by more efficient ones.

GeneXus: Incremental Development Made True

GeneXus applies this theory.

GeneXus is a tool based on the "user views"; it captures their knowledge and systematizes them in a knowledge base. From its knowledge base, GeneXus is capable of designing, generating and maintaining in a fully automatic way the structure of the database and the application programs (the programs required for the users to be able to operate with their views).

GeneXus is built on solid mathematical principles.

This is the main strength of GeneXus: **an excellent management of the knowledge of business systems**.

GeneXus works with pure knowledge, which permits to do several things: to generate programs (traditional software), understand the knowledge of human beings (it does not need additional knowledge –which would never be updated), and to automatically operate that knowledge (by integrating it to other knowledge coming from other sources, spreading it, granting licenses to third parties to incorporate it to their applications). In short, GeneXus enables the "**business of knowledge**", as a step forward in relation to the "**business of software**".

Another advantage of working with pure knowledge is the possibility of generating applications for multiple platforms and multiple architectures and, in particular, being able to have some kind of "insurance" for technological changes: for example, the users of GeneXus that developed applications 8 or 10 years ago for AS/400 with text screens and quite primitive technologies, can now take advantage of the knowledge on the development of these applications that GeneXus saved to develop Java and/or .NET applications easily, although when these applications were developed, nobody could think of anything as different from the environment on which they worked.

When an application is developed with GeneXus the first stage is the **Design** that is done by recording the views of users (based on which the system captures and systematizes knowledge).

Next comes the **Prototype** stage in which GeneXus generates the database (structure and data) and the programs for the prototype environment. Once the Prototype has been generated it must be tested by the analyst and the users.

If improvements and errors are detected during the Prototype test, we return to the Design phase, the required changes are made and then we go back to the Prototype. We call this cycle the Design/Prototype cycle.

Once the Prototype has been approved, we go to the **Implementation** phase, where GeneXus generates, also automatically, the database and the programs for the production environment.

In short, an application starts with a Design, then a Prototype, it is then Implemented or put in the production stage and at any of the previous stages it may be sent back to Design to make changes.

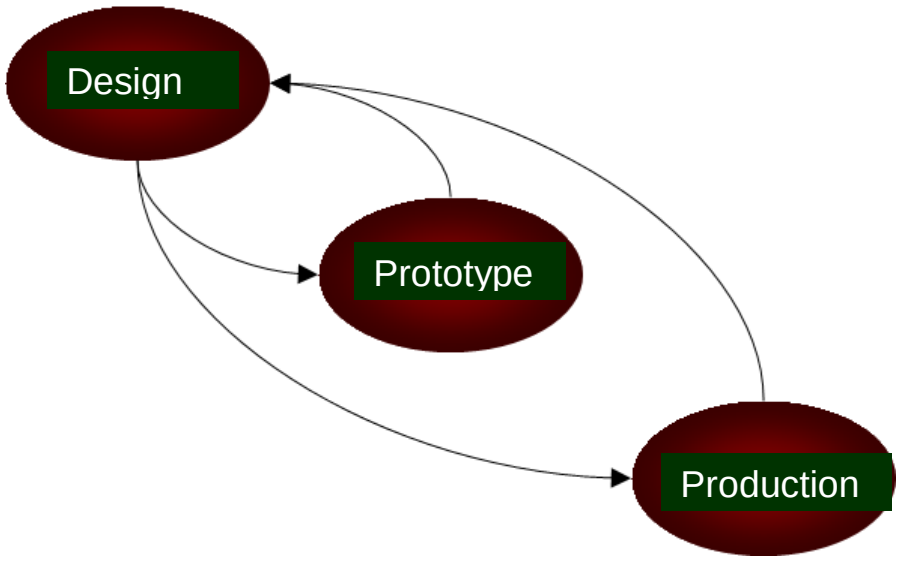

Figure 1 - Design-Prototype and Design-Production Cycles

These tasks are described below.

Design

This task is performed jointly by the analyst and the user and it consists in identifying and describing the data views of users.

The work is done in the user's environment. This method permits to work with a low level of abstraction, using terms and concepts that are familiar to the final user.

A very important consequence of this is that the user's attitude is highly participatory. The system becomes a joint work and as the user follows its evolution constantly, quality is much better than usual.

As we have already mentioned, GeneXus captures knowledge using views of objects from the user's reality. The types of objects supported by GeneXus are, among others: **Transactions, Reports, Procedures, Work Panels, Web Panels, Data Views, and BI Transactions**.

The design task consists mainly in identifying and describing these objects. Using these descriptions and in automatic way, GeneXus systematizes the knowledge captured and starts to build, in an incremental way, the Knowledge Base.

This Knowledge Base is a single repository of all the design information, using which GeneXus creates the physical data model (tables, attributes, tables of contents, redundancies, referential integrity rules, etc.), and the application programs.

Therefore, the main task in the analysis and design of an application is centered on the description of GeneXus objects.

The following are the most important types of GeneXus objects:

Transactions

A transaction is an interactive process or a screen (Win or Web) that permits users to create, modify, or eliminate information from the database.

Examples:

- A screen to create, modify or delete Company Customers.
- A billing screen: a process that enables a user to create bills and print them.

A screen enables the user to perform different actions such as inserting, updating, deleting, printing without the need to return to the main menu.

The transaction has essential elements such as the screen data structure, the rules of the business, and formulas and cosmetic elements such as the format of screens (in this case

the developer may use the editors available to format it or choose to use the one automatically inferred by the system).

Reports

A report is a process that permits to view data from the database. The list output may be sent to a screen or to a printer (and in this case we have a conventional list).

This object may be used to define simple lists (for example, a list of customers) or highly sophisticated lists, with several control cuts, multiple readings from the database and parameterization.

A report may not update the database.

It also has a GXquery tool to perform dynamic reports on the database. For more information see: www.genexus.com/gxquery

Procedures

This object has all the features of Reports and also permits to update the database. Procedures are commonly used for two types of processes:

- *Batch updating processes*. For example: deleting all bills paid prior to a certain date
- *General use subroutines.* For example: a written amount routine where, given an amount a literal is retrieved with the amount in words (1010 => 'A thousand and ten')
- *Processes to be run on an application server or database server*: processes (in general written on Java or.NET) for Multi Tier Architecture, to be run on an application or database server.

Work Panels

A Work Panel is a screen that enables the user to perform interactive database queries. The more users use their computers for their work, the higher the need for using sophisticated dialogs that permit them to sit in front of the computer and think. Work Panels enable the design of this type of user dialogs.

For example: A Work Panel that shows the list of customers and permits the user to choose to view customers' bills or pending accounts.

Web Panels

These are similar to the Work Panels group but require the use of an application navigator (Browser) to be run on Internet/Intranet/Extranet environments.

Data Views

These permit to establish correspondences between preexisting database tables and GeneXus tables and to operate them with the same intelligence as GeneXus objects.

Knowledge-based Development

Using the objects described above, the physical data model is designed based on the Relational Database Theory and ensuring a normal third form database (without redundancy). This normalization is carried out automatically by GeneXus.

However, the analyst may define redundancies, which are then automatically managed (controlled or propagated, as applicable), by GeneXus.

GeneXus' repository maintains the design specifications in an abstract way; that is, it does not depend on the target environment, which allows generating equivalent functional applications from the same repository to be run on different platforms.

Multiple Platforms / Multi Tier Architecture

As a consequence of the above, it is possible, for example, for a user of a centralized IBM AS/400 application, 100% developed with GeneXus, maybe 15 years ago, to operate it fully or in part on a JAVA or .NET environment without the need to modify the original objects.

In the last few years, generating multi-platform applications has become necessity, that is, executing the same application on several environments. For example, a banking system application must be capable of running on an iSeries or Linux server in the central office and on a PC net in the bank branches.

But that is not all; with the increasing use of Client/Server and Internet/Intranet/Extranet environments, a new need has emerged: the same application must have one of its parts running on a certain platform and others running on other platforms. In these cases, it is also essential to have a correct intercommunication among the different parts of the platform.

Developing applications on GeneXus also provides the possibility of dividing an application in such a way that each part may be executed on a different platform, using the most appropriate language to generate programs on each one of these platforms. This has lead to the development of multiple tier architectures, which also optimize the use of the resources available, and new technologies.

Prototyping

Design tasks carry the difficulties of all human communication:

- The user fails to remember certain details.
- The analyst fails to record certain elements.
- The user makes some wrong assessments.
- The analyst misinterprets some of the user explanations.

But the implementation of systems is also generally a time consuming task, therefore:

- Because many of these problems are detected only at the final testing stage of the system, the cost of solving them (both in time and money) is very high.
- Reality is always changing; therefore, it is not reasonable to think that specifications may be frozen when the system is implemented.
- The consequence of freezing specifications is that the finally implemented solution is relatively unsatisfactory.

The impact of these problems would be greatly diminished if it were possible to test each specification immediately, and to know the repercussion of each change on the rest of the system.

A first approach to this, offered by several systems, is the possibility of showing the user screen formats, reports, etc., animated by menus. This helps the user have an idea of the system being built, but it always has surprises later.

An entirely different situation would be to make available to the user for execution an application functionally equivalent to the desired one, up to the smallest detail.

This is what GeneXus does: **A GeneXus prototype is a complete application, functionally equivalent to the production application.**

The difference between prototyping and production is that the former is performed on a microcomputer environment, while production is performed on the target environment of the user (IBM iSeries, Linux server, Client / Server, JAVA, .NET, iOS, Android, BlackBerry, etc.). The prototype permits to fully test the application before production. During these tests, the final user may work with real data, testing in a natural way, not only the screen formats, reports, etc., but also the formulas, business rules, data structures, etc.

The philosophy of GeneXus is based on the concept known as **incremental development**. When working on a traditional environment, the changes made to the project during implementation, and in particular once the system is implemented, are very expensive (and rarely well documented). GeneXus solves this problem: it builds the application using a successive approach methodology that permits, once the need for changes has been detected, the user to prototype them and test them immediately, without any additional cost.

Implementation

GeneXus automatically generates the code required to:

- Create and maintain the database;
- Generate and maintain the programs to manage the objects described by the user.

The generation process can be divided in two stages: SPECIFICATION and GENERATION. Specification is fully independent form the target environment, but generation is not. This means that it is possible to execute the same model on the different execution platforms for which it has been generated, and each one of the versions generated may be optimized for the environment on which it will be run.

The most important environments and languages supported to the date of this document (see cover) are:

Platforms

Execution platforms

JAVA, Microsoft .NET, Microsoft .NET Compact Framework

Operative Systems

IBM OS/400, LINUX, UNIX, Windows NT/2000/2003 Servers, Windows NT/2000/XP/CE and Windows Vista

Internet

JAVA, ASP.NET, Visual Basic (ASP), C/SQL, HTML, Web Services

Mobile

iOS, Android, BlackBerry

Databases

IBM DB2 for iSeries and UDB, Informix, Microsoft SQL Server, MySQL, Oracle and PostgreSQL

Languages

JAVA, C#, COBOL, RPG, Visual Basic

Web Servers

Microsoft IIS, Apache, WebSphere, etc.

Multiple Architectures

Multiple tier architectures, web-based, Client/Server, centralized (iSeries), mobile

For a complete list of the technologies currently supported go to:
http://www.genexus.com/technologies

GeneXus also offers a set of supplementary tools for:

- Workflow – GXflow (www.gxflow.com)
- Reporting – GXquery (www.gxquery.com)
- Business Intelligence – GXplorer (www.gxplorer.com)
- Portal Building – GXportal (www.gxportal.com)
- GeneXus Server (http://www.genexus.com/gxserver)
- Gxtest (http://www.genexus.com/gxtest)

Maintenance

One of the most important features of GeneXus, which clearly sets it apart from competitors, is that the maintenance of both the database (structure and contents) and the programs is fully automatic.

The following is an explanation of the maintenance process when introducing changes to the description of any GeneXus object (user view):

Impact of Changes on the Database

Impact Analysis

Once the user view changes have been described, GeneXus automatically analyzes their impact on the database a produces a report explaining how to make the data conversion and, if applicable, the potential problems of such conversion (inconsistencies due to old data against new rules, etc.). The analyst decides whether to accept the impact and continue or not.

Generation of Conversion Programs

Once the problems have been solved or the default conversion proposed by GeneXus has been accepted, the programs used to perform the conversion (structure and content) from the old database to the new one are automatically generated.

Execution of Conversion Programs

The next step is to go to the corresponding execution environment (prototype, Internet production, Client / Server production, etc.) and execute the conversion programs.

Impact of Changes on Programs

Impact Analysis

GeneXus analyzes the impact of changes on programs and generates a diagnosis reporting on the programs that must be generated or re-generated and also providing either the navigation diagram or a pseudo-code for the new program, as chosen by the analyst

Generation of New Programs

The system subsequently generates or re-generates all the programs automatically.

Documentation

All the knowledge supplied by the analyst, or inferred by GeneXus, is available from an active repository that constitutes a very complete, constantly updated online documentation.

The documentation includes the description of specific objects and information on the resulting knowledge base and the database designed.

The knowledge base of GeneXus not only permits to access the knowledge stored anytime the developer wants it, but also enables access to all the information logically inferred (a referential integrity rule, a navigation map on the database, an analysis of the impact of changes, cross references, E-R diagrams inferred from the knowledge stored, etc.).

Consolidation of Several Applications and Re-utilization of Knowledge

Several applications may be designed and prototyped at the same time, by different teams, using GeneXus. These teams may exchange design specifications using GeneXus Knowledge Manager.

This module permits to perform the following tasks automatically:

- Start the design of a new application based on Business Objects, Software Patterns, Domains, Attributes and/or Styles from a public domain (see: http://www.gxopen.com.uy).
- Distribute knowledge from a corporate knowledge base to the knowledge base of another application.
- Verify the concordance between the knowledge base of an application and the corporate knowledge base.
- Consolidate two applications (this is especially useful to consolidate knowledge from a given application to the corporate knowledge base).

This provides an ideal flexibility: the analyst works freely on a prototype environment, with a small knowledge base and, only when the application is ready from the point of view of the user, the corporate knowledge base, which is generally very large, needs to be taken into account.

At this point, with powerful automatic aids, the impact of the new application is determined, or the modification of the preexisting one and, if applicable, the changes needed to ensure consistency are performed in a very simple way.

With this scheme it is possible to re-use, for example, licensed Knowledge Bases from third parties.

At the beginning, it is necessary to use a common nomenclature for the different knowledge bases involved in the consolidation. However, the "Adapt From" functionality permits to define a mapping for the conversion of names to adapt to the target nomenclature.

It is also important to point out that the software house granting the license on the GeneXus knowledge base may maintain the confidentiality of some of its parts, thus permitting its automatic use without disclosing its sources.

Additionally, it is possible for an object to be labeled as public or private. All may be used automatically by GeneXus, but in the case of private objects, only the owner may see and/or modify the high level source of GeneXus.

It is important to highlight that GeneXus has a feature that permits to generate applications in several languages on the same knowledge base, which helps greatly in the international use of applications.

Unique Features of GeneXus

GeneXus has some unique features that set it apart from competitors. Among others:

- The design is based on the views supplied by users. Because of their daily activities, they are the ones who know how things should and should not work.
- The description of each object is fully independent, so that, in case of needing to modify the description of one, this will not imply the need to manually modify the

description of any other. This exclusive feature of GeneXus (ortogonality of descriptions) is the one that permits a fully automatic maintenance of applications.
- The learning curve is short.
- Fully automatic design, creation and maintenance of databases.
- The application (databases and programs) is always high quality, regardless of the modifications it undergoes:
    - The database is always optimum (third normal form).
    - No programs are modified: when programs are no longer adequate, new optimum ones, not amended, are generated to replace them.
- Use of preexisting files or databases as GeneXus own.
- Powerful, very high level languages for the definition of Processes, Work Panels and Web Objects. The descriptions of processes in these languages are performed without referring to the files involved, which are automatically inferred at generation time. This feature enables full independence of data from such specifications. Consequently, GeneXus high level specifications do not need to be modified when modifying the database.
- 100% automatic maintenance: All of these elements permit GeneXus to automatically generate and maintain a 100% of the programs on business applications (commercial, administrative, financial, industrial, etc.).
- GeneXus works on PCs, leaving the production environment totally free for the processing of applications.
- Easy distribution of corporate knowledge to facilitate the development of new applications.
- Simple and powerful Reporting and Data Warehousing solutions.
- Automatic consistency checks and consolidation across applications developed separately.
- Platform and architecture independence.
- Simplicity: GeneXus uses the most advanced artificial intelligence resources for the analyst and the users to use in a very simple way.

Who Are the Users of GeneXus?

Over 5,000 customers, with over 40,000 licenses throughout the world, use GeneXus to create and integrate mission critical applications that adapt easily to the inevitable changes of the business. The technology of GeneXus permits customers to use its exclusive business know-how on the leading technological platforms of the market.

Corporate customers include medium-sized to very large companies in a wide range of industries. They currently account for 65% of the billing of GeneXus.

Software houses include small, medium-sized and large software companies that build their solutions using GeneXus technology. This segment currently accounts for 35% of billing, and is rapidly growing.